Improving Disk I/O Performance on Linux

Carl Henrik Lunde, Håvard Espeland, Håkon Kvale Stensland, Andreas Petlund, Pål Halvorsen
Completely Fair Queuing

Default scheduler on Linux

Ensures complete fairness among I/O-requests in the same class by assigning time slices to each process

Provides some level of QoS by assigning processes to different classes.
Completely Fair Queuing

Gives relatively high throughput by waiting for close requests from the same process (spatial locality)

Latency is kept proportional to system load by scheduling each process periodically

Work-conserving scheduler

If the active process is considered idle, CFQ will prematurely end the time slice, store the residual time, and move on to the next process.
What happens at the different layers

<table>
<thead>
<tr>
<th>Blockdev</th>
<th>CPU</th>
<th>Sequence</th>
<th>Time</th>
<th>PID</th>
<th>Event</th>
<th>Sector + blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,48</td>
<td>3</td>
<td>0</td>
<td>2.004027488</td>
<td>8593</td>
<td>m</td>
<td>N enter vfs_read</td>
</tr>
<tr>
<td>8,48</td>
<td>3</td>
<td>657</td>
<td>2.004047581</td>
<td>8593</td>
<td>A</td>
<td>R 831107423 + 256 &lt;- (8,49)3831107360</td>
</tr>
<tr>
<td>8,48</td>
<td>3</td>
<td>658</td>
<td>2.004047704</td>
<td>8593</td>
<td>Q</td>
<td>R 831107423 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>3</td>
<td>659</td>
<td>2.004048664</td>
<td>8593</td>
<td>G</td>
<td>R 831107423 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>3</td>
<td>660</td>
<td>2.004049761</td>
<td>8593</td>
<td>I</td>
<td>R 831107423 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>3</td>
<td>661</td>
<td>2.004155593</td>
<td>8593</td>
<td>A</td>
<td>R 831107679 + 256 &lt;- (8,49)3831107616</td>
</tr>
<tr>
<td>8,48</td>
<td>3</td>
<td>662</td>
<td>2.004155696</td>
<td>8593</td>
<td>Q</td>
<td>R 831107679 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>3</td>
<td>663</td>
<td>2.004156385</td>
<td>8593</td>
<td>M</td>
<td>R 831107679 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>3</td>
<td>664</td>
<td>2.004157215</td>
<td>8593</td>
<td>U</td>
<td>N [ iosched - bench ] 7</td>
</tr>
<tr>
<td>8,48</td>
<td>2</td>
<td>599</td>
<td>2.010858781</td>
<td>8598</td>
<td>C</td>
<td>R 33655111 + 256 [0]</td>
</tr>
<tr>
<td>8,48</td>
<td>2</td>
<td>600</td>
<td>2.010983458</td>
<td>8598</td>
<td>A</td>
<td>R 33655623 + 256 &lt;- (8,49)33655560</td>
</tr>
<tr>
<td>8,48</td>
<td>2</td>
<td>601</td>
<td>2.010983593</td>
<td>8598</td>
<td>Q</td>
<td>R 33655623 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>2</td>
<td>602</td>
<td>2.010984420</td>
<td>8598</td>
<td>G</td>
<td>R 33655623 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>2</td>
<td>603</td>
<td>2.010985094</td>
<td>8598</td>
<td>I</td>
<td>R 33655623 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>2</td>
<td>604</td>
<td>2.010988047</td>
<td>8598</td>
<td>D</td>
<td>R 33655623 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>2</td>
<td>605</td>
<td>2.010990666</td>
<td>8598</td>
<td>U</td>
<td>N [ iosched - bench ] 7</td>
</tr>
<tr>
<td>8,48</td>
<td>1</td>
<td>530</td>
<td>2.012061299</td>
<td>8598</td>
<td>C</td>
<td>R 33655367 + 256 [0]</td>
</tr>
<tr>
<td>8,48</td>
<td>1</td>
<td>531</td>
<td>2.012182151</td>
<td>8598</td>
<td>A</td>
<td>R 33655879 + 256 &lt;- (8,49)33655816</td>
</tr>
<tr>
<td>8,48</td>
<td>1</td>
<td>532</td>
<td>2.012182705</td>
<td>8598</td>
<td>Q</td>
<td>R 33655879 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>1</td>
<td>533</td>
<td>2.012183893</td>
<td>8598</td>
<td>G</td>
<td>R 33655879 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>1</td>
<td>534</td>
<td>2.012184605</td>
<td>8598</td>
<td>I</td>
<td>R 33655879 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>1</td>
<td>535</td>
<td>2.012187585</td>
<td>8598</td>
<td>D</td>
<td>R 33655879 + 256 [ iosched - bench ]</td>
</tr>
<tr>
<td>8,48</td>
<td>1</td>
<td>536</td>
<td>2.012190294</td>
<td>8598</td>
<td>U</td>
<td>N [ iosched - bench ] 7</td>
</tr>
<tr>
<td>8,48</td>
<td>3</td>
<td>665</td>
<td>2.012834865</td>
<td>8598</td>
<td>C</td>
<td>R 33655623 + 256 [0]</td>
</tr>
</tbody>
</table>

(...
Performance of I/O-schedulers

To evaluate I/O-schedulers, we wrote a tool to visualize the behaviour.

I/O requests are captured using `blktrace`, and we can analyse what happens at each level in great detail.
CFQ Behaviour Visualized

S₃ 14.3/16 MiB/s 8192 KiB
S₂ 15.6/16 MiB/s 8192 KiB
S₁ 8.0/8 MiB/s 4096 KiB
S₀ 8.1/8 MiB/s 4096 KiB
R₁ 39.0/40 KiB/s 4 KiB
R₀ 40.0/40 KiB/s 4 KiB
M F₀ 388.0 KiB/s 446 files
Suggested improvements

Add a top level elevator for better performance
Suggested improvements

Improve the seeking application

Queuedepth=1  I/O scheduler has no choices!
CFQ

Current best-effort proportional share “QoS” is not sufficient for applications with bandwidth requirements

How many MiB/s is 200 ms/s?

Real-time priority can starve other processes

Filling buffers can leave other processes with a missed deadline

Hard to do work conservation in RT class
Adding QoS to CFQ

Introduce a bandwidth class that can coexist with other CFQ classes

Reserves bandwidth in terms of bytes per second in contrast to proportional disk time

Do not change behaviour of existing classes

Priority

- Real-Time (RT)
- Bandwidth (BW)
- Best Effort (BE)
- BW Work Conserving
- Idle
CFQ Bandwidth Class Proof-of-Concept

Implemented as Token Bucket to allow bursty behaviour, like VBR video streaming

Avoid seeks by introducing a top-level elevator for BW readers which improves global throughput

Work conservation enables BW readers to utilize the disk in excess of the reserved bandwidth if no best effort-readers are pending

Unused reserved bandwidth are distributed among best effort readers
Evaluation of BW class

Benchmarks use videos in a streaming scenario

- HD video at 3 MiB/s (25 Mbit/s)
- SD video at 1 MiB/s

In some tests additional greedy readers consumes data at a rate faster than the disk can provide.
Deadline Misses

Find the maximum number of reserved readers that we can add while still maintaining deadlines

Evaluate effect of simultaneous best-effort readers

“Background noise” readers

10 HD stream readers in Bandwidth class

3 random BE, reading 4 KB 20 times per second

5 sequential BE, reading 4 MiB/s
# Deadline Misses

<table>
<thead>
<tr>
<th>Streams</th>
<th>1 MiB/s</th>
<th>3 MiB/s</th>
<th>CFQ BW SSF</th>
<th>CFQ BW C-SCAN</th>
<th>RT</th>
<th>AS</th>
<th>Deadline</th>
<th>noop</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>479</td>
<td>345</td>
<td>281</td>
</tr>
<tr>
<td>17</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>482</td>
<td>330</td>
<td>398</td>
</tr>
<tr>
<td>18</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>492</td>
<td>300</td>
<td>783</td>
</tr>
<tr>
<td>19</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>512</td>
<td>276</td>
<td>1061</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>179</td>
<td>505</td>
<td>288</td>
<td>1123</td>
</tr>
<tr>
<td>21</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>186</td>
<td>531</td>
<td>381</td>
<td>1075</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>276</td>
<td>517</td>
<td>947</td>
<td>1061</td>
</tr>
<tr>
<td>23</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>549</td>
<td>1233</td>
<td>1064</td>
</tr>
<tr>
<td>24</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>553</td>
<td>1276</td>
<td>1054</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>182</td>
<td>188</td>
<td>N/A</td>
<td>N/A</td>
<td>536</td>
<td>1279</td>
<td>1033</td>
</tr>
</tbody>
</table>

Deadline misses with 1 s deadline / round
Disk head movement

- CFQ RT (no elevator)
- CFQ BW SSF
- CFQ BW C-SCAN
Isolation

Best-effort readers should not affect reserved readers

Greedy reserved readers should not get more bandwidth than reserved if it would affect any best effort reader
Work Conservation of BW class

<table>
<thead>
<tr>
<th>Reserved</th>
<th>Requested</th>
<th>Result</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>$\infty$</td>
<td>63.5 MiB/s</td>
<td>138 ms max (31 ms avg)</td>
</tr>
<tr>
<td>Video A</td>
<td>BE4</td>
<td>8 MiB/s</td>
<td>676 ms max (48 ms avg)</td>
</tr>
<tr>
<td>Video B</td>
<td>BE4</td>
<td>8 MiB/s</td>
<td>8.0 MiB/s</td>
</tr>
</tbody>
</table>

Two Best Effort streams, requesting 8 MiB/s each

One BW class stream with 32 MiB/s requested

The reserved reader requests as much as possible, and receives more than the reservation allows.
Optimizing Multi-file Performance

The I/O-scheduler can not improve the multi-file reader.

Only a limited number of requests to choose from before deadline expires

How fast would it be without disk seeking?
Simulation

Created an utility to reorder traces of disk activity

Sort requests by block number to minimize seeks

Dispatch requests without delay (ignore CPU-time)

Replay trace on real hardware
Simulation of tar

GNU Tar archiving the Linux source (~ 22 500 files) on ext4

Without seeks (read as single file)

One Sweep reorder

GNU Tar (unmodified)

Seconds
Result of simulation

Potential improvement in applications that accesses many files on rotational disks

For example: cp, zip, rsync, tar, scp, rm, find, dpkg (database), Tracker, File Manager and many others.

Proposed solution: Order I/O requests relative to physical location on disk before they hit the kernel scheduler
Userspace I/O Scheduling

We know the relative location of meta data by using inode number (at least on ext{3,4})

\[
\text{inode}(a) < \text{inode}(b) \Rightarrow \text{block}(\text{inode}(a)) \leq \text{block}(\text{inode}(b))
\]

On ext4, any user can ask for position of file data

FIEMAP ioctl

Use this knowledge to avoid seeks
Implementation

First pass:

Read all meta data ordered by inode number (inodes and directories)

New directories are discovered during traversal, so we cannot sort all meta data in advance

Use a C-SCAN elevator for meta-data

Second pass:

Read file data ordered by file data position
Improved tar results

GNU Tar archiving the Linux source on ext4 (~ 22 500 files)

- Without seeks (read as single file)
- One Sweep reorder
- Improved tar (qtar)
- GNU Tar (unmodified)

Seconds
Visualization of seek footprint (ext4)

(a) GNU Tar

(b) Improved tar
Filesystem Aging

Archiving the Linux kernel (22 500 files) on ext4
Video

Video of GNU Tar vs. modified tar (qtar)

Aged ext3 file system

Around 5000 files

Some read errors, the disk is dying...

Notice the two passes of the qtar algorithm
Conclusion

Visualizing blktraces can be very useful

CFQ benefits from a top-level elevator

QoS should be available for programs with bandwidth requirements

A simple bandwidth class added to CFQ works fairly well

The kernel I/O scheduler can only optimize requests that are pending. Ordering requests in user space mitigates this.
Questions?